HW3

1. (20pt)

   a. *Construct the divided-difference table from these data:*

x	$f(x)$	Δf	$\Delta^2 f$	$\Delta^3 f$	$\Delta^4 f$
-0.2	1.23	2.22	-11.88	-103.58	73.61
0.3	2.34	-8.48	-1.53	-81.5	
0.7	-1.05	-7.56	14.78		
-0.3	6.51	-16.43			
0.1	-0.06				

   b. *Use the divided-difference table to interpolate for $f(0.4)$ with the first three points.*

$$P_2(x) = 1.23 + 2.22(x + 0.2) - 11.88(x + 0.2)(x - 0.3)$$

$$P_2(0.4) = 1.849$$

   c. *Repeat (b) but use the best set of three points. Which points should be used?*

We first choose three points closest to 0.4 for better accuracy, which are 0.3, 0.7 and 0.1.

Construct the divided-difference table based on these sorted points:

x	$f(x)$	Δf	$\Delta^2 f$
0.3	2.34	-8.48	-34.125
0.7	-1.05	-1.65	
0.1	-0.06		

Then we can use the divided-difference table to interpolate for $f(0.4)$.

$$P_2'(x) = 2.34 - 8.48(x - 0.3) - 34.125(x - 0.3)(x - 0.7)$$

$$P_2'(0.4) = 2.52$$

2. (20pt) *Fit the function below with a natural cubic spline that matches to $f(x)$ at five evenly spaced points in $[-1, 1]$. Use end conditions 3 and 4 to plot the spline curve together with $f(x)$. Which end condition gives the best fit to the function?|*

$$f(x) = \begin{cases} 0 & \text{for } -1 < x < -0.5 \\ 1 - 2|x| & \text{for } -0.5 \le x \le 0.5 \\ 0 & \text{for } 0.5 < x < 1 \end{cases}$$

In this problem, I utilized the `CubicSpline` function from Python's `scipy.interpolate` module to plot both the natural cubic spline and the cubic spline with condition 4. However, the module does not offer built-in support for condition 3. To address this, I calculated the function for each section as follows:

First, modify $HS = Y$ based on condition 3:

$$\begin{bmatrix} 3h_0 + 2h_1 & h_1 & 0 \\ h_1 & 2(h_1 + h_2) & h_2 \\ 0 & h_2 & 2(h_2 + h_3) \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} = 6 \begin{bmatrix} f[x_1, x_2] - f[x_0, x_1] \\ f[x_2, x_3] - f[x_1, x_2] \\ f[x_3, x_4] - f[x_2, x_3] \end{bmatrix}$$

And the divided-difference table are following:

i	x_i	$f(x_i)$	$f[x_i, x_{i+1}]$	h_i
0	-1	0	0	0.5
1	-0.5	0	2	0.5
2	0	1	0	0.5
3	0.5	0	0	0.5
4	1	0		

Based on the divided-difference table, we can obtain the following equation:

$$\begin{bmatrix} 2.5 & 0.5 & 0 \\ 0.5 & 2 & 0.5 \\ 0 & 0.5 & 2 \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 12 \\ -24 \\ 12 \end{bmatrix} \Rightarrow \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} 8 \\ 8 \\ -16 \\ 8 \\ 8 \end{bmatrix}$$

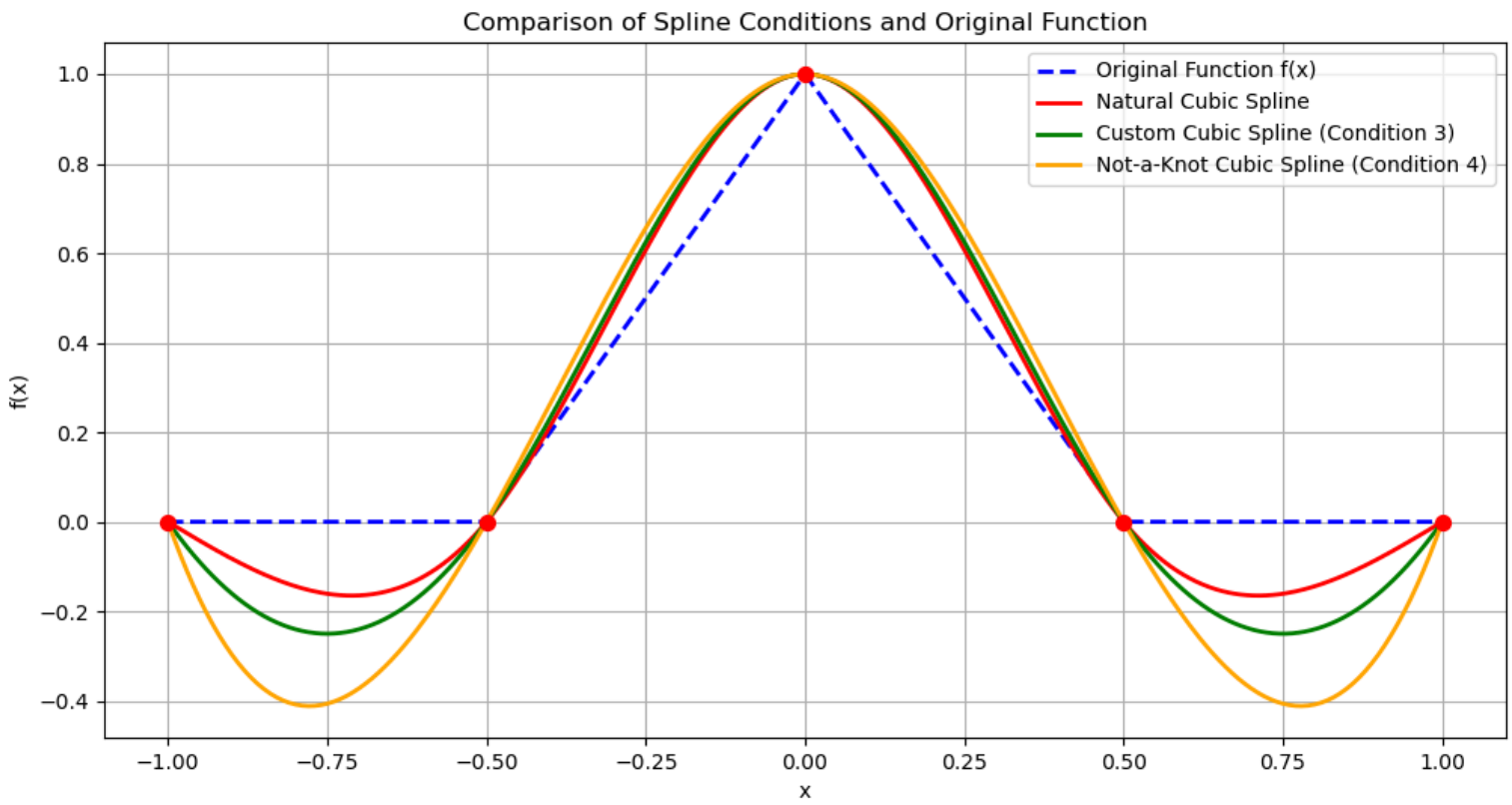
By the definition of a, b, c, d:

$$a_i = \frac{S_{i+1} - S_i}{6h_i},\ b_i = \frac{S_i}{2},\ c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{2h_iS_i - h_iS_{i+1}}{6},\ d_i = y_i$$

We can obtain the following table and each cubic spline function.

i	h_i	S_i	a_i	b_i	c_i	d_i	$g_i(x)$
0	0.5	8	0	4	-2	0	$4(x+1)^2 - 2(x+1)$
1	0.5	8	-8	4	2	0	$-8(x+0.5)^3 + 4(x+0.5)^2 + 2(x+0.5)$
2	0.5	-16	8	-8	0	1	$8x^3 - 8x^2 + 1$
3	0.5	8	0	4	-2	0	$4(x-0.5)^2 - 2(x-0.5)$
4		8					

Result:

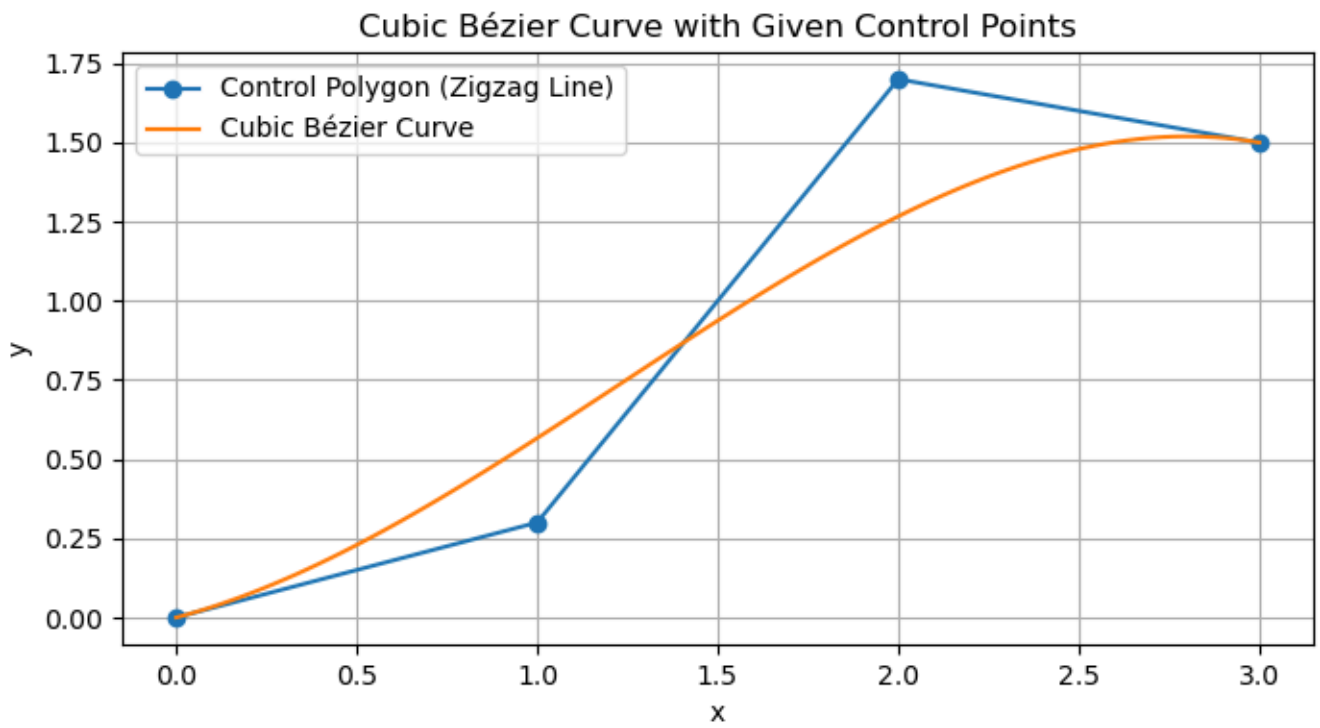


As illustrated in the diagram below, the natural cubic spline provides the best fit to the function, with condition 3 yielding better results than condition 4.

3. (20pts) *If these four points are connected in order by straight lines, a zigzag line is created:*

$$(0, 0), (1, 0.3), (2, 1.7), (3, 1.5).$$

   a. *Using the two interior points as controls, find the cubic Bezier curve. Plot this together with the zigzag line*



b. *If the second and third points (the control points) are moved, the Bezier curve will change. If these are moved vertically, where should they be located so that the Bezier curve passes through all of the original four points?*

To solve for the specific positions of the control points (point two and three) that make the cubic Bézier curve pass through all four given points, I used the Bézier equation:

$$\mathrm{B}(t) = (1-t)^3 \cdot \mathrm{P}_0 + 3(1-t)^2 \cdot t \cdot \mathrm{P}_1 + 3(1-t) \cdot t^2 \cdot \mathrm{P}_2 + t^3 \cdot \mathrm{P}_3$$

Here, P_0 and P_3 are the endpoints of the curve, while P_1 and P_2 are the control points we need to adjust. The given points through which the curve must pass were:

- $\mathrm{P}_0 = (0, 0)$ at $t = 0$
- $\mathrm{P}_3 = (3, 1.5)$ at $t = 1$
- Two additional points:
  - $(1, 0.3)$ at $t = 1/3$
  - $(2, 1.7)$ at $t = 2/3$

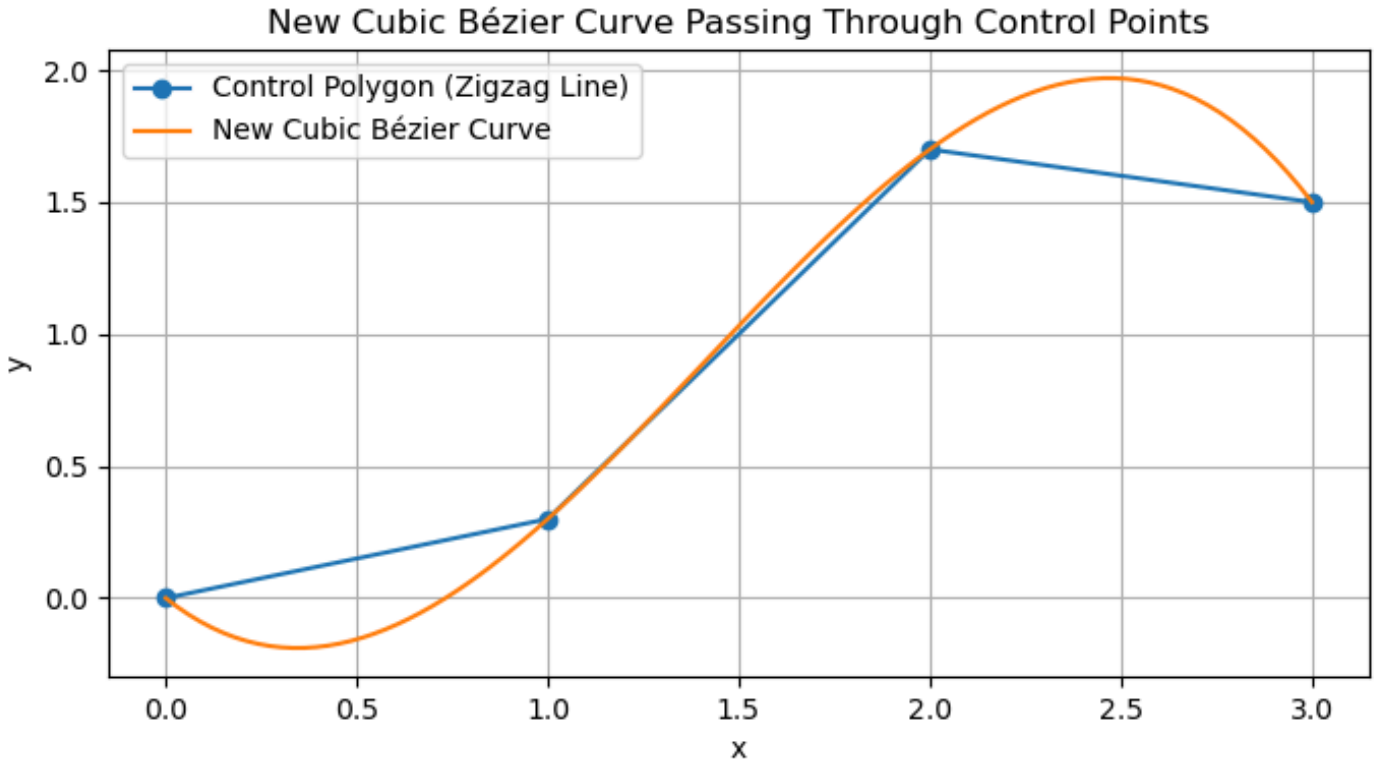
I set up the equations to ensure that the curve passes through these midpoints at $t = 1/3$ and $t = 2/3$, then solved for the coordinates of P_1 and P_2.

Let $\mathrm{P}_1 = (1, y_1)$ and $\mathrm{P}_2 = (2, y_2)$, then we can have:

$$\begin{cases} 0.3 = 4/9\ y_1 + 2/9\ y_2 + 1.5 \times 1/27 \\ 1.7 = 2/9\ y_1 + 4/9\ y_2 + 1.5 \times 8/27 \end{cases}$$

Then we can solve that:

$$y_1 = -1.15,\ y_2 = 3.4$$



4. (20pts) *The function whose values are tabulated below is z = x + ey. Construct the B-spline surface from the rectangular array of 16 points nearest to (2.8, 0.54) and find z(2.8, 0.54).*

$x \backslash y$	0.2	0.4	0.5	0.7	0.9
1.3	2.521	2.792	2.949	3.314	3.760
2.5	3.721	3.992	4.149	4.514	4.960
3.1	4.321	4.592	4.749	5.114	5.560
4.7	5.921	6.192	6.349	6.714	7.160
5.5	6.721	6.992	7.149	7.514	7.960

I ran the following python code to help me find z(2.8, 0.54):

```
import numpy as np
from scipy.interpolate import RectBivariateSpline

# Define the grid points and their corresponding z values from the problem statement
x = np.array([1.3, 2.5, 3.1, 4.7, 5.5])
y = np.array([0.2, 0.4, 0.5, 0.7, 0.9])
z = np.array([
    [2.521, 2.792, 2.949, 3.314, 3.760],
    [3.721, 3.992, 4.149, 4.514, 4.960],
    [4.321, 4.592, 4.749, 5.114, 5.560],
    [5.921, 6.192, 6.349, 6.714, 7.160],
    [6.721, 6.992, 7.149, 7.514, 7.960]
])

# Create the B-spline surface
spline = RectBivariateSpline(x, y, z)

# Interpolate to find the value at (2.8, 0.54)
z_value = spline(2.8, 0.54)

print("The interpolated value of z(2.8, 0.54) is:", z_value[0][0])
```

Output:

```
The interpolated value of z(2.8, 0.54) is: 4.51628224
```

5. (20pts) *The equation of a plane is z = ax + by + c. We can fit experimental data to such a plane using the least-squares technique. Here are some data for z = f(x, y)*

x	0.40	1.2	3.4	4.1	5.7	7.2	9.3
y	0.70	2.1	4.0	4.9	6.3	8.1	8.9
z	0.031	0.933	3.058	3.349	4.870	5.757	8.921

a. *Develop the normal equations to fit the (x, y) data to a plane.*

b. *Use these equations to fit z = ax + by + c.*

c. *What is the sum of the squares of the deviations of the points from the plane?*

In this set of problem, I ran the following python code to help me find out a, b, c and calculate the sum of squares of deviations.

```
import numpy as np

# Define the data points
x = np.array([0.40, 1.2, 3.4, 4.1, 5.7, 7.2, 9.3])
y = np.array([0.70, 2.1, 4.0, 4.9, 6.3, 8.1, 8.9])
z = np.array([0.031, 0.933, 3.058, 3.349, 4.870, 5.757, 8.921])

# Assemble the matrix A with an additional column of ones for the intercept
A = np.vstack([x, y, np.ones(len(x))]).T

# Use least squares to solve for a, b, and c
coefficients, residuals, rank, s = np.linalg.lstsq(A, z, rcond=None)

# Coefficients are a, b, and c
a, b, c = coefficients

# Calculate the fitted z values
z_fitted = a * x + b * y + c

# Calculate the sum of squares of residuals
sum_of_squares = np.sum((z - z_fitted) ** 2)

print(f"Plane equation: z = {a:.3f}x + {b:.3f}y + {c:.3f}")
print(f"Sum of squares of deviations: {sum_of_squares:.3f}")
```

Output:

```
Plane equation: z = 1.596x + -0.702y + 0.221
Sum of squares of deviations: 0.319
```

6. (20pts) *Find the first few terms of the Chebyshev series for cos(x) by rewriting the Maclaurin series in terms of the T(x)'s and collecting terms. Convert this to a power series in x. Compare the error of both the Chebyshev series and the power series after truncating each to the fourth degree.*

The Maclaurin series for $\cos(x)$ is:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots$$

Express each power of x in term of Chebyshev polynomials, for simplicity and given the question, we'll approximate up to the fourth degree:

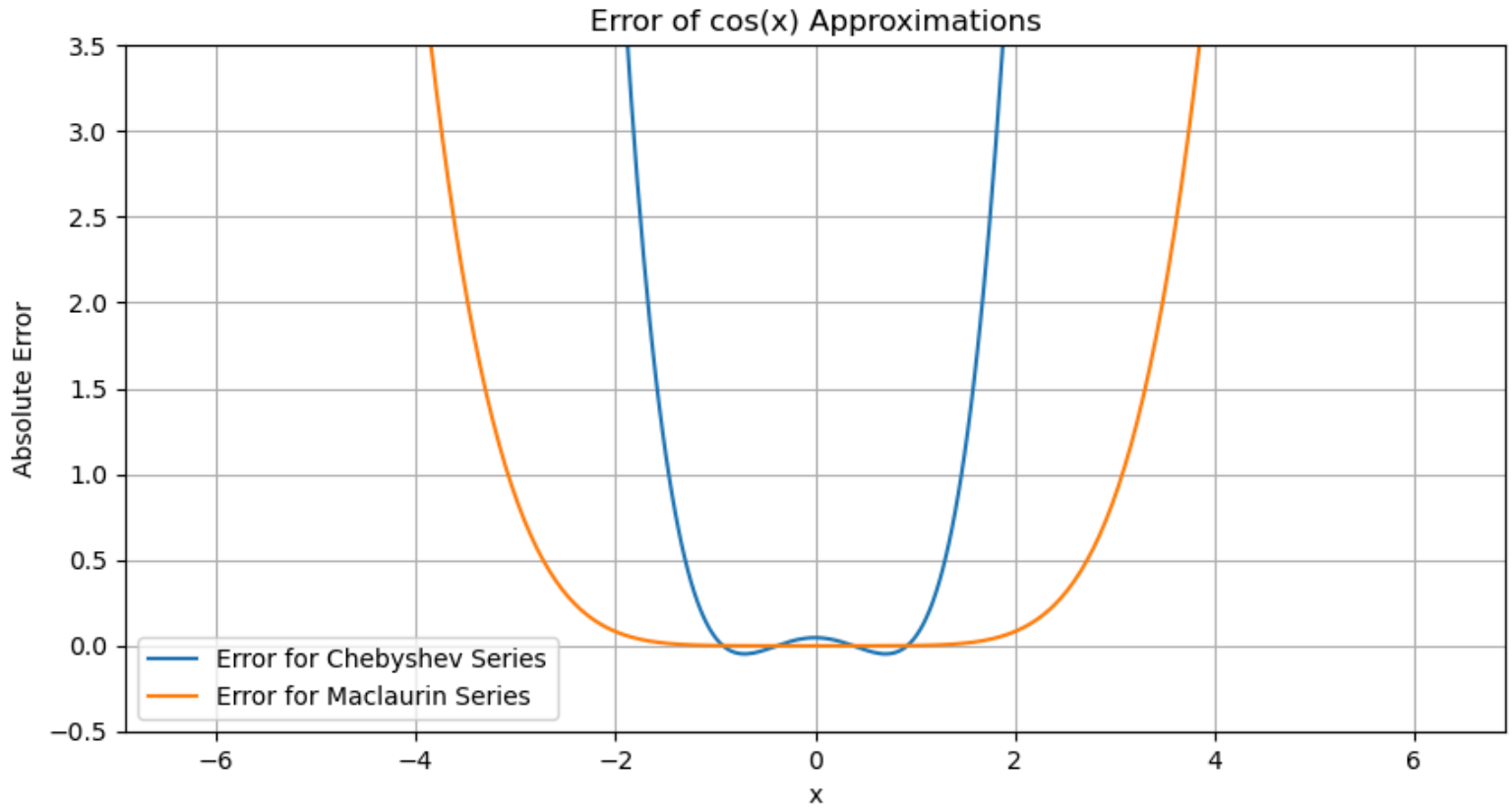
$$x^2 \approx \frac{1}{2}(T_2(x) + T_0(x))$$
$$x^4 \approx \frac{1}{8}(T_4(x) + 4T_2(x) + 3T_0(x))$$

Substituting these approximations back into the Maclaurin series gives:

$$\cos(x) \approx 1 - \frac{1}{2}\left(\frac{1}{2}(T_2(x) + T_0(x))\right) + \frac{1}{24}\left(\frac{1}{8}(T_4(x) + 4T_2(x) + 3T_0(x))\right)$$

Simplifying, and collecting terms for $T_0(x), T_2(x)$, and $T_4(x)$:

$$\begin{aligned}\cos(x) &\approx \left(1 - \frac{1}{4} + \frac{1}{64}\right)T_0(x) + \left(-\frac{1}{4} + \frac{1}{48}\right)T_2(x) + \frac{1}{192}T_4(x) \\ &\approx 0.7656T_0(x) - 0.2292T_2(x) + 0.0052T_4(x) \\ &= 1.0474 - 0.8794x^2 + 0.4211x^4\end{aligned}$$



The result show that error for Chebyshev series is bigger than that for Maclaurin series. This is because the function $cos(x)$ is already very well approximated by its Maclaurin series in the interval [–1,1].

7. (20pts)*Find the Fourier coefficients for $f(x) = x^2 - 1$ if it is periodic and one period extends from $x = -1$ to $x = 2$.*

```
from sympy import symbols, integrate, cos, sin, pi, N

# Define variables
x, n = symbols('x n', real=True)

# Function definition
f = x**2 - 1

# Period
P = 3

# Define the integrals for coefficients
a_0 = (2/P) * integrate(f, (x, -1, 2))
a_n_expr = (2/P) * integrate(f * cos(2 * pi * n * x / P), (x, -1, 2))
b_n_expr = (2/P) * integrate(f * sin(2 * pi * n * x / P), (x, -1, 2))

# Evaluate the integrals for n = 1 to 4
print("a_0 =", N(a_0))

for ni in range(1, 5):
```

```
    a_n = N(a_n_expr.subs(n, ni))
    b_n = N(b_n_expr.subs(n, ni))
    print(f"a_{ni} =", round(a_n,4))
    print(f"b_{ni} =", round(b_n,4))

print("a_n =", a_n_expr.simplify())
print("b_n =", b_n_expr.simplify())
```

Output:

```
a_0 = 0
a_1 = -1.2829
b_1 = -0.3123
a_2 = 0.2995
b_2 = 0.4362
a_3 = 0.1013
b_3 = -0.3183
a_4 = -0.2352
b_4 = 0.0700
a_n = Piecewise(((3.0*pi**2*n**2*sin(4*pi*n/3) + 3.0*pi*n*(cos(2*pi*n/3) + 2*cos(4*pi*n/3)) - 4.
5*sin(2*pi*n/3) - 4.5*sin(4*pi*n/3))/(pi**3*n**3), (n > 0) | (n < 0)), (0, True))
b_n = Piecewise(((-3.0*pi**2*n**2*cos(4*pi*n/3) - 3.0*pi*n*(sin(2*pi*n/3) - 2*sin(4*pi*n/3)) -
4.5*cos(2*pi*n/3) + 4.5*cos(4*pi*n/3))/(pi**3*n**3), (n > 0) | (n < 0)), (0, True))
```